WEB DEVELOPMENT USING MERN STACK

Version Control System

Git vs GitHub

a tool to keep track of all the changes we do in our project, also known as Source Code Management Tool

it makes "save points" that saves our project

provides total development freedom

easy file recovery

keep track of commit (who introduced an issue and when)

rollback feature (previous working state)

example - Git, subversion, perforce

Git is a version control tool and GitHub is a service that hosts git repositories

a special folder on which Git can watch

~~Local Version Control System~~ → ~~Centralised Version Control System~~ → Distributed Version Control System ✓✓

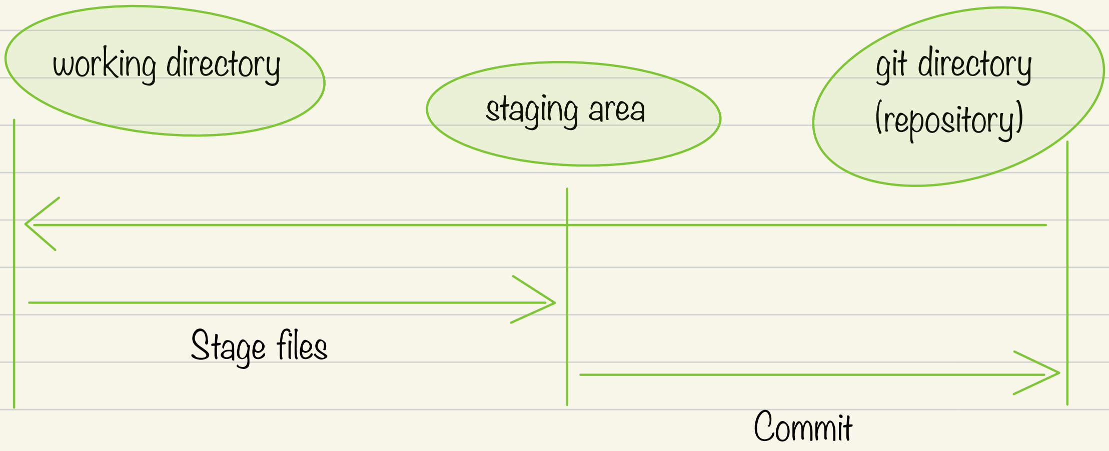
Installing git on macOS

1. Open Terminal
2. Check if Git is present using this cmd : `git --version`
3. If not present then, go to browser and type : git-scm.com
4. For macOS, we will use these cmds:
    1. `brew install git` (Homebrew tool won't be there so, install brew
    2. to install Homebrew :

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

    3. if asked, input the password and continue the installation
    4. Run left steps to add brew to environment variables, possibly 3 cmds
    5. Now finally run this cmd : `brew install git`
5. Git will be installed
6. To check, run cmd : `git --version` and we will see git version ... ...

Git 3 Stage Architecture

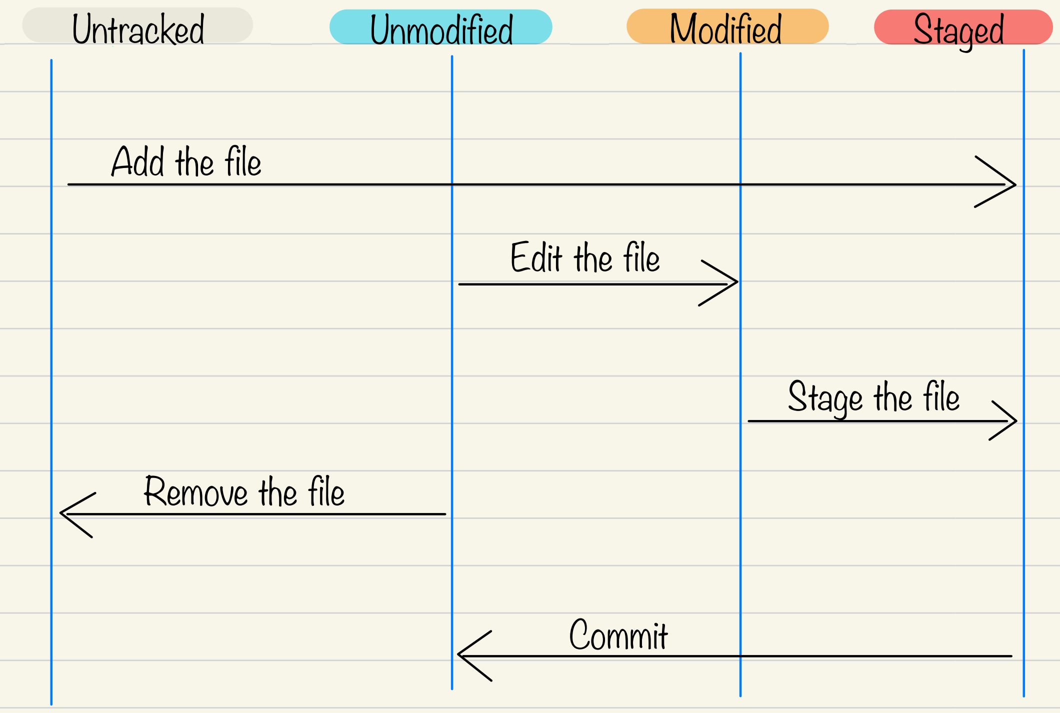


Git features

1. Captures snapshot not difference
2. Almost every operation is local
3. Git has integrity (no random can come and make changes to our file and then reflect it to our folder. Git generates SHA1 checksum for every change

Secure Hash Algorithm 1

Different stages of a file in git



Creating a git repo

1. To initialise a folder to a Git repository, use this cmd : git init
   Now, the folder is a Git repo with a by default hidden .git file. If we want to see hidden folder, use this shortcut : cmd + shift + . Or if we want to see the same hidden folder on terminal, use this cmd : ls-al
2. Use this cmd to check status of repo :
   git status
3. To clone a Git repo, copy the url of that git repo and run this cmd in terminal :
   git clone <url> <new folder name in your local>/
4. To commit the files, use these cmds :
   1. to stage/add all the files : git add --a
      OR
      to stage/add any specific file : git add <file>
   2. now, to commit : git commit -m "<message>"

.git folder consists of metadata of git repo

cmd to check what's modified : git diff

cmd to generate files : touch <file name>

Review a repo history

1. If we want to see log of commits, use this cmd :
   `git log`
2. If we want to see log of commits along with modified changes i.e. diff, then use this cmd :
   `git log -p`
3. If we want to see log of commits and not the modified changes but the files that are modified then use this cmd :
   `git log --stat`
4. Use this cmd for only commit ids and commit messages in one line :
   `git log --oneline`
5. To see what are the changes in any particular commit id, then use this cmd :
   `git show <commit id>`
6. To discard any change, use this cmd :
   all files : `git restore "."`
   specific file : `git restore <filename>`

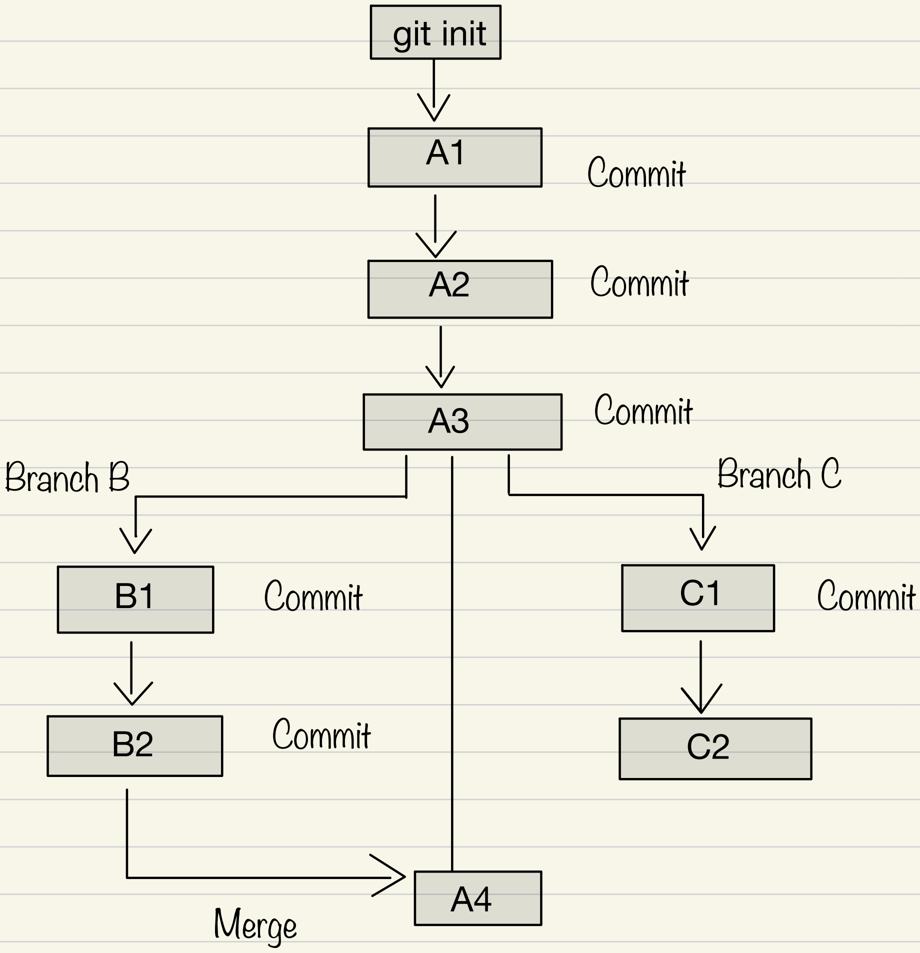
Use this cmd to see latest "n" number of commits :
`git log -n`

7. to make any file untracked, we can keep those files in a special file named ".gitignore"

Example : if we want any .txt files to remain untracked, we just need to open .gitignore file and mention *.txt and then all txt files will be ignored

Branching, Tagging, Merging

Branching



1. To know name of all the branches, use this cmd :

   ```
   git branch
   ```

2. To make a new branch out of master or any other branch, use this cmd :

   ```
   git branch <new branch name>
   ```

3. To switch to any specific branch, use this cmd :

   ```
   git switch <branch name in which to switch>
   ```

   OR

   ```
   git checkout <branch name in which to switch>
   ```

4. To make a new branch as well as switch to the new branch, use this cmd :

   ```
   git checkout -b <new branch name>
   ```

5. To add and commit with the same cmd :

   ```
   git commit -a -m "<commit message>"
   ```

   All tracked files, staged files will go to commit BUT if we have an untracked file, it will not go to commit using this cmd, we have to add it to staging specifically

6. To merge the changes of a specific branch to master branch :

   1. First, move to master branch and then take latest :

      ```
      git pull
      ```

   2. then, use this cmd :

      ```
      git merge <branch name to merge>
      ```

7. To delete branch, use this cmd :

   `git branch -d <branch name to delete>`

8. To tag a specific commit, use this cmd :

   `git tag -a <tag name> <commit id> -m "<commit message>"`

9. To delete a specific tag, use this cmd :

   `git tag -d <tag name to delete>`

10. To stash the local code changes, git add . and then :

    `git stash`

11. To unstash the local changes :

    `git stash apply`

merge conflict may arise if the local stash changes conflict at any line with the latest code changes

12. To finally push the code after commit :

    `git push`

    OR

    `git push -u origin master`

Undo commits

1. To amend the most latest commit i.e. alter the commit message in latest commit :

   `git commit --amend`

   Note : after this cmd runs, we will have the option to change the commit message if we want. Press i first to enable editing of commit message. Once done press esc button then : then w then q . DONE .

2. Revert given commit using commit id :

```
git revert <commit id>
```

3. To delete commit (most dangerous cmd) :

```
git reset --soft <commit id to where we want to reach>
```

4. To unstage any specific file which is staged by mistake using git add command, we can use this cmd :

```
git restore --staged <file name to unstage>
```

To connect our project to remote repository :

```
git remote add origin <url from github for remote repo>
```

5. If somehow, our files get messed up and we want our code back with no mishappenings, then we can match our all files to the last commit :

```
git checkout -f
```

6. To delete a file from working directory as well as staging area, use this cmd :

```
git rm <filename to delete>
```

7. To remove a file only from the staging area, sort of UNSTAGE, use this cmd :

```
git rm --cached <filename to unstage>
```